APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: **APPLICATION DEVELOPMENT METHOD**

APPLICANT: **Norbert Hoffmann**
Steve Briscoe
Phil Main

"EXPRESS MAIL" Mailing Label Number EL478578591US

Date of Deposit January 3, 2002

I hereby certify under 37 CFR 1.10 that this correspondence is being deposited with the United States Postal Service as **"Express Mail Post Office to Addressee"** with sufficient postage on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Melissa Scantillo

Melissa Scantillo

APPLICATION DEVELOPMENT METHOD

CROSS-REFERENCE TO RELATED APPLICATION

This application claims the benefit of the filing date of U.S. Provisional Application serial number 60/275,884 entitled "METHOD AND SYSTEM FOR AUTOMATING APPLICATION DEVELOPMENT" which was filed on March 14, 2001.

BACKGROUND

The following invention relates software development and, in particular, to a method for efficiently developing and maintaining business software.

Organizations spend considerable resources in developing and maintaining computer system software to meet the business needs of the organization. Typically, the process of developing business software begins with the ultimate users of the software specifying the functions the software is to perform and communicating those specifications to software developers. The software developers then take these functional requirements and write program code that, when compiled into an executable program, will perform the specified functions.

Often, the software developers will have to rewrite portions of the program code in order to meet the needs of the user. In some cases, the program code originally produced by the software developers may not have met the requirements as specified by the user and thus may require modification. In other situations, the user may have changed the original specification thereby causing the software developers to have to make modifications accordingly. In either case, the software developers typically modify portions of the previously written program code so that the resulting software program meets the requirements of the user.

The prior art process of developing business applications is fraught with inefficiencies. The requirements for business applications are typically identified and specified by the business users who then communicate these requirements to the software developers. During this process, there are often miscommunications and misunderstandings between the business users and the software developers as to the precise functional requirements the software program is to meet. As a result, the software developers may have to make numerous revisions to the program code in order to fully satisfy the requirements of the business user. Aside from the time and costs associated with undergoing numerous revision cycles, the resulting program code is often difficult to maintain as a result of the many changes made by the software developers. Furthermore, as these changes are continuously made, the original design specification produced by the business users no longer accurately documents the actual operation of the final version of the business application.

Additional inefficiencies arise in the development of business applications in a large organizations that employ numerous business applications. Large organizations typically form several software development teams that each perform the software development for different groups of business users. In many cases, the development projects of some of the software development teams have common requirements such as, for example, communicating with various computer systems. It is difficult, however, to coordinate the different development teams so that program code needed to satisfy the common requirements is shared between the teams. What often occurs is that each team separately writes program code to perform these common requirements which necessarily increases the development and maintenance costs of the business applications.

Another significant drawback of having several teams develop business software applications in a large organization is that often different business applications need to communicate with each other in order to achieve an organization-wide purpose. For example, in a financial institution, a software application designed to execute a trade request of a client typically communicates with a software application that settles the trade and confirms the trade to client. If each of the business applications developed by different development teams are designed using different software development methodologies, considerable modifications are often required to have the different business applications properly communicate with each other.

Accordingly, it is desirable to provide a method and system for the efficient development and maintenance of business software.

SUMMARY OF THE INVENTION

The present invention is directed to overcoming the drawbacks of the prior art. Under the present invention a method is provided for developing an application and includes a step of selecting one of a plurality of patterns wherein each of the plurality of patterns have at least one of a plurality of business artifacts. Next, the application is designed using the at least one of the plurality of business artifacts associated with the selected one of the plurality of patterns. Next, code based on the at least one of the plurality of business artifacts is generated. Finally, the code is interfaced with at least one platform independent service.

In an exemplary embodiment, the plurality of patterns includes a process workflow application pattern, a service request application pattern, a web-based application pattern and a reporting pattern.

In another exemplary embodiment, the plurality of business artifacts includes a process business artifact, an activity business artifact, a user interface business artifact, a business object business artifact and a data business artifact.

In yet another exemplary embodiment, the plurality of patterns include a process workflow application pattern and the at least one of the plurality of business artifacts associated with the process workflow application pattern includes a process business artifact, an activity business artifact, a user interface business artifact, a business object business artifact and a data business artifact.

In still yet another exemplary embodiment, the plurality of patterns include a service request application pattern and the at least one of the plurality of business artifacts associated with the service request application pattern includes an activity business artifact, a business object business artifact and a data business artifact.

In an exemplary embodiment, the plurality of patterns include a web-based application pattern and the at least one of said plurality of business artifacts associated with the web-based application pattern includes a user-interface business artifact, a business object business artifact and data business artifact.

In another exemplary embodiment, the plurality of patterns include a reporting pattern and the at least one of said plurality of business artifacts associated with the reporting pattern includes a user interface business artifact and a data business artifact.

In yet another exemplary embodiment, a plurality of business artifact services are included and includes the step of interfacing the code with the at least one platform independent service via the business artifact services.

In still yet another exemplary embodiment, the business artifact services includes a workflow framework, an activity framework, a user interface services, a business object framework and a database.

In an exemplary embodiment, the platform independent services include logging services, security services, messaging services and transaction services.

In another exemplary embodiment the at least one platform independent service is interfaced with a platform specific adapter.

In yet another exemplary embodiment, the platform specific adapter is selected from a group including an NT adapter, a Solaris adapter and a S390 adapter.

In still yet another exemplary embodiment, a workflow using said at least one of the plurality of business artifacts is formed.

In an exemplary embodiment, the application is modified by changing at least one of the plurality of business artifacts associated with the selected one of the plurality of patterns and regenerating code based on the at least one of the plurality of business artifacts.

In another exemplary embodiment, the code is converted into an executable format.

Under the present invention, a system for developing an application is provided and includes a modeling engine for selecting one of a plurality of patterns, each of the plurality of patterns having at least one of a plurality of business artifacts, and for designing the application using the at least one of the plurality of business artifacts associated with the selected one of the plurality of patterns. Also included is a forward engineering module for generating code based on the at least one of the plurality of business artifacts wherein the code interfaces with at least one platform independent service.

In an exemplary embodiment, a plurality of business artifact services are included and the code interfaces with the at least one platform independent service via the business artifact services.

In another exemplary embodiment, the at least one platform independent service interfaces with a platform specific adapter.

In yet another exemplary embodiment, a workflow is formed using the at least one of the plurality of business artifacts.

In still yet another exemplary embodiment, the application is modified by changing at least one of the plurality of business artifacts associated with the selected one of the plurality of patterns and the code based on the at least one of the plurality of business artifacts is regenerated.

In an exemplary embodiment, the forward engineering module converts the code into an executable format.

Under the present invention, a method for developing an application is provided and includes a means for selecting one of a plurality of patterns, each of the plurality of patterns having at least one of a plurality of business artifacts; a means for designing the application using the at least one of the plurality of business artifacts associated with the selected one of the plurality of patterns; a means for generating code based on the at least one of the plurality of business artifacts; and a means interfacing the code with at least one platform independent service.

Accordingly, a method and system is provided for the efficient development and maintenance of business software.

The invention accordingly comprises the features of construction, combination of elements and arrangement of parts that will be exemplified in the following detailed disclosure,

and the scope of the invention will be indicated in the claims. Other features and advantages of the invention will be apparent from the description, the drawings and the claims.

DESCRIPTION OF THE DRAWINGS

For a fuller understanding of the invention, reference is made to the following description taken in conjunction with the accompanying drawings, in which:

FIG. 1 is a block diagram illustrating a workflow application pattern according to the present invention;

FIG. 2 is a block diagram illustrating a web-based application pattern according to the present invention;

FIG. 3 is a block diagram illustrating a reporting application pattern according to the present invention;

FIG. 4 is a block diagram illustrating a service request application pattern according to the present invention;

FIG. 5 is a flowchart of the process by which a business application is developed according to the present invention;

FIG. 6 is a block diagram of a system for developing business applications according to the present invention; and

FIG. 7 is an activity diagram for checking the validity of an address according to an exemplary embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention provides a method for efficiently developing and maintaining business software applications. Central to the method of the present invention is the use of

application patterns. An application pattern is a template used for building a software application that implements a solution to a particular business problem. In an exemplary embodiment various application patterns exist with each being suitable for building a different class of business applications. Because the application patterns of the present invention formalizes the development of business applications, as will be described below, the use of applications patterns enables an organization to generate business applications quickly and consistently across the entire organization.

Application patterns consist of business artifacts that are the building blocks with which business applications are constructed. The number and type of business artifacts that are included in a particular application pattern depends on the type of business application the particular application pattern is designed to solve.

Referring now to FIG. 1, there is shown a block diagram illustrating a workflow application pattern 100. Workflow application pattern 100 is suitable for building mission-critical business applications that are characterized by a high-degree of repetition and automation and that require a high-volume, high-reliability and high-throughput transaction processing capability. In order to build such business applications, workflow application pattern 100 uses the following five business artifacts: a process business artifact 103, an activity business artifact 105, a user-interface business artifact 107, a business object business artifact 109 and a data business artifact 111.

Process business artifact 103 is a workflow process definition that describes the mechanism by which a business operation is to be carried out. More specifically, process business artifact 103 describes a number of activities that are to be used to solve a particular business problem and a number of rules that govern the operation flow between those activities.

In an exemplary embodiment, a workflow process definition may be constructed using a software tool called Rational Rose (http://www.rational.com/products/rose/index.jsp) for visually modeling the activities and rules that form a workflow process definition.

Once a workflow process definition associated with process business artifact 103 is formed, the workflow process definition is received by a workflow framework business artifact service 113 that transforms the workflow process definition into software code that forms the code skeleton of the desired business application. Referring now to FIG. 7, there is shown an activity diagram 701 for checking the validity of an address that may be part of a workflow process definition, according to an exemplary embodiment. In addition to a start node 703 and a stop node 705, activity diagram 701 includes a ValidateAddress automatic activity 707, an AddressValid automatic activity 709 and an AddressInvalid automatic activity 711. Also included in activity diagram 701 is a FixInvalidAddress manual activity 713 and a CreateConfirmation subprocess 715. In an exemplary embodiment, workflow framework business artifact service 113 receives activity diagram 701 and generates the code example contained in Exhibit A of the Appendix. Workflow framework 113 may forward engineer a workflow process definition into any type of software code suitable for developing a business application including, by way of non-limiting example, Java code. The process of transforming a workflow process definition into software code may be performed using Together ControlCenter from TogetherSoft (http://www.togethersoft.com/products/controlcenter/index.jsp;jsessionid=mtfwcen1t1.www6).

Activity business artifacts 105 are the individual business functions that are included in the workflow process definition that are required solve the particular business problem and represent the lowest level of decomposition within a workflow process definition. In an

exemplary embodiment, activity business artifacts 105 include both automatic activities as well as manual activities that require user intervention. In an automated activity, the functions associated with the activity is performed by software whereas in a manual activity the work is performed by a user of the business application. In an exemplary embodiment, activities may change between being automatic and manual, for example, when the volume of a manual activity increases sufficiently to make it cost effective to be automated.

An activity may be specified to any level of granularity as desired. In an exemplary embodiment, the level of granularity should be selected to maximize the ease by which the workflow in the process definition can be changed and increase the opportunity to re-use the activity in different business applications. For example, in the context of a trading system, a finely grained activity to validate an instrument that is a subject of a trade may be applicable across all trade and product types. However, a coarser grained activity to validate a trade may require logic for specific product types (for e.g., equity and fixed income) and therefore may not be useable across different the product types.

In an exemplary embodiment, manual activities are presented to the user through a user-interface (to be described below) that presents the user with what is needed to complete the work to be done by the business application. For example, in the context of a trading system, a manual activity may require the user to handle an exception (for e.g., something wrong with the trade details) or provide an authorization (for e.g., release of a free of payment instruction).

Once the functions associated with activity business artifact 105 are specified, the functions are received by an activity framework business artifact service 115 that transforms the functions' specifications into software code sections that perform these functions. These code sections may include the middleware required to access any of a plurality of platform

independent services 117 such as, by way of non-limiting example, security services 117(a), messaging services 117(b) and transaction services 117(c). These code sections interface with the code skeleton generated by workflow framework 113, as required. Exhibit B of the Appendix shows the generated code sections that interface with the code skeleton of Exhibit A associated with activity diagram 601, according to an exemplary embodiment. Activity framework 115 may forward engineer a functional description of an activity into any type of software code suitable for developing a business application including, by way of non-limiting example, Java code.

User-interface business artifact 107 is a mechanism by which users interact with a business application. For example, user-interface business artifact 107 may enable a business user to view and manually resolve exceptions that may prevent the fully automated execution of a workflow process.

Once the mechanism associated with user-interface business artifact 107 is specified, the specification is received by a UIS services business artifact service 119 that generates the necessary codes sections to implement the specified user interaction. These code sections also interface with the code skeleton generated by workflow framework 113, as required. In an exemplary embodiment, UIS services 119 may generate code sections for implementing a graphical user-interface via which a user performs manual activities in connection with the operation of the business application, as required. Exhibit C of the Appendix includes an exemplary code section for a user interface for the FixInvalidAddress manual activity 613 included in activity diagram 601.

Business object business artifact 109 is a software entity that encapsulates functional logic and data that is used by the business application and that includes a programmatic interface

by which operations may be performed on a business object. Activity business artifacts 105 are carried out by performing operations on business object business artifacts 109.

Once the business logic and data associated with business object business artifacts 109 is specified, the specification is received by a business object factory business artifact service 121 that generates the necessary codes sections to retrieve the required data and perform the specified business logic on the retrieved data. These code sections also interface with the code skeleton generated by workflow framework 113, as required. Exhibit D of the Appendix includes a code section to retrieve data and perform specified business logic on the retrieved data, according to an exemplary embodiment.

Finally, data business artifacts 111 are the individual pieces of information that are encapsulated within business object business artifacts 109 and upon which the business logic contained in business object business artifacts 109 is performed. The data is stored in a database business artifact services 123 such as, by way of non-limiting example, a DB2 database server.

Once the relevant code sections for the particular application are generated, the application is initiated via workflow routing software that invokes the codes sections as required. Exhibit E of the Appendix is an exemplary workflow routing software that invokes the relevant classes and methods for a particular workflow application.

An example of a workflow application pattern that uses the five business artifacts is a securities settlement system that includes a process (workflow) of settling trades (business objects) that were executed by a trading system. The process includes tasks that are automatically performed by the system (automatic activities) that may include, for example, determining the account balance of the party executing a particular trade. The process may also

include a manual activity in which a user overseeing the operation of settlement system must contact the trading party to resolve any discrepancy in the trading party's account information.

Platform independent services 117 interfaces with any of a plurality of platform adapters 125 so that the resulting business application may execute on and interface with a corresponding operating environment. Platform adapters 125 may include, by way of non-limiting example, adapters for interfacing the business application to a Windows NT platform, an IBM S390 platform and a Solaris platform.

Referring now to FIG. 2, there is shown a block diagram illustrating a web-based application pattern 200. Elements that are similar to elements included in the embodiment of FIG. 1 are identically labeled and a detailed description thereof is eliminated.

Web-based application pattern 200 is used for building business applications that retrieve and display business data to a web-based front-end as well as store to a database updated or newly entered data received via the front-end. In order to build business applications that conform to web-based application pattern 200, only user-interface business artifact 107, business object business artifact 109 and data business artifacts 111 are required. Because these business applications do not include a workflow and activities being performed within a workflow, web-based application pattern 200 does not require process and activity business artifacts. Exhibit F of the Appendix includes an example of a user-interface business artifact, business object business artifact and data artifact associated with a web-based application for fixing an invalid address.

Once the desired business artifacts are selected, the specified business artifacts are received by UIS services business artifacts services 119, business object business artifacts services 121 and data business artifacts services 123, respectively, for generating the code

sections that result in the desired business application. These code sections interface with platform independent services 117, as required. Furthermore, because process and activity business artifacts are not used in business applications that conform to web-based application pattern 200, code sections that are associated with those business artifacts are eliminated from the resulting code that forms the business application.

Referring now to FIG. 3, there is shown a block diagram illustrating a reporting application pattern 300. Elements that are similar to elements included in the embodiment of FIG. 1 are identically labeled and a detailed description thereof is eliminated.

Reporting application pattern 300 is used for building applications in which information is delivered to any output device such as, by way of non-limiting example, a web-browser, a printer or an application program (for example, an Excel spreadsheet). In an exemplary embodiment, the reporting application pattern includes tools such as, by way of non-limiting example, style sheets that may be used for conforming reports to any desired look and feel. In order to build business applications that conform to reporting application pattern 300, only user-interface business artifact 107 and data business artifact 111 are used. Exhibit G of the Appendix includes an example of a user-interface business artifact and data artifact associated with a reporting application for responding to transaction inquiries.

Once the desired business artifacts are selected, the specified business artifacts are received by UIS services business artifacts services 119 and data business artifacts services 123, respectively, for generating the code sections that result in the desired business application. These code sections interface with platform independent services 117, as required. Furthermore, because process, activity and business object business artifacts are not used in business applications that conform to reporting application pattern 300, code sections that are associated

with those business artifacts are eliminated from the resulting code that forms the business application.

Referring now to FIG. 4, there is shown a block diagram illustrating a service request application pattern 400. Elements that are similar to elements included in the embodiment of FIG. 1 are identically labeled and a detailed description thereof is eliminated.

Service request application pattern 400 is used for building business applications in which a two systems directly communicate with each other. An example of such a business application is a settlement system requesting from a trading system information regarding executed trades and the trading system replying to the settlement system with the requested information. In order to build business applications that conform to service request application pattern 400, activity business artifact 105, business object business artifact 109 and data business artifact 111 are used. Exhibit H of the Appendix includes an example of activity business artifacts, business object business artifacts and data artifacts associated with a configuration service request application.

Once the desired business artifacts are selected, the specified business artifacts are received by activity framework business artifacts services 115, business object factory business artifact services 212 and data business artifacts services 123, respectively, for generating the code sections that result in the desired business application. These code sections interface with platform independent services 117, as required. Furthermore, because process and user-interface business artifacts are not used in business applications that conform to service request application pattern 400, code sections that are associated with those business artifacts are eliminated from the resulting code that forms the business application.

Referring now to FIG. 5, there is shown a flowchart of the process by which a business application is developed according to the present invention. Initially, in Step 501 the developer selects the type of application pattern that is suited for solving a particular business problem. Next, in Step 502, for each of the business artifacts types used by the selected application pattern, the developer specifies the business artifacts that perform the functional requirements of the business application. Once the business artifacts are specified, then in Step 503, the code for the business application is forward engineered based on the specified business artifacts. Next, in Step 504, the code is compiled thereby producing an executable business application that solves the particular business problem.

Next, in Step 505, it is determined whether the developer desires to make modifications to the resulting business application. Modifications may be made for any reason including either because the business artifacts as originally specified do not precisely solve the business problem or because the business problem to be solved has changed. In any case, if it is desired to modify the business application, the method returns to Step 502 in which case the business artifacts are specified again. The method then continues to Steps 503 and 504 in which the code underlying the business application is forward engineered and converted into an executable business application. It is important to note that according to the method of the present invention, any modifications made to a business application are performed by re-specifying the business artifacts and re-forward engineering the code for the business application. Modifications to the business application are not made, however, by making changes to the code directly. If changes are made to the code directly, then the functionality of the resulting business application will differ from the business application specification as embodied by the business artifacts. By requiring that modifications be made by re-specifying business artifacts, however, the

functionality of the resulting business application is accurately described and documented by the specified business artifacts.

Referring now to FIG. 6, there is shown a block diagram of a system 600 for developing business applications according to the present invention. Included in system 600 is a modeling engine 603 that is accessed by a user operating a user access device (for e.g., a personal computer). Modeling engine 603 provides the user with the tools for specifying business artifacts and designing business applications using business artifacts in accordance with the methods described above. Once the user has specified and designed the business application using modeling engine 603, the design is forwarded by modeling engine 603 to a forward engineering module 605 that converts the modeled design into software code and outputs an executable business application that accurately reflects the design provided by the user.

In an exemplary embodiment, modeling engine 603 and forward engineering module 606 of system 600 are a software program executing on a computer that performs the functions described above.

Accordingly, under the present invention a method and system is provided for efficiently developing and maintaining business software applications. Because the business applications created using the present invention are developed using visual modeling tools, the applications may be completely specified by the business people that actually use the application. Also, the business users can easily redesign the business application by making changes to the specified business artifacts using the modeling tools. As a result, inefficiencies that typically arise from miscommunications between business users and software developers are eliminated.

Furthermore, because the business logic that represents the unique functionality of the business application is generated by the business people using modeling tools, the business logic

is separately identifiable and not buried within the application code. As a result, the application model, that contains the specified business artifacts and upon which the generated application code is based, serves to accurately document the functionality of the resulting business application.

In addition, by structuring applications using the types of business artifacts that are required to support a particular application type, legacy business applications can be effectively re-engineered by breaking up their implementation into reusable workflows, activities, and business objects according to the methods of the present invention.

Yet another benefit of developing business applications using the method of the present invention is that the platform and infrastructure upon which the business applications run is generic and does not contain code that is specific to any particular business application. Accordingly, a large percentage of a business application developed according the present invention consists of application code that is common to other business applications developed using these methods. Business applications are therefore distinguished solely on the business logic and business artifacts specific to the particular business application. Furthermore, because business applications shared a common code structure, the connectivity between such business applications and between the business applications and operating platforms and systems are greatly simplified.

A number of embodiments of the present invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Based on the above description, it will be obvious to one of ordinary skill to implement the system and methods of the present invention in one or more computer programs that are executable on a programmable system including at least one programmable

processor coupled to receive data and instructions from, and to transmit data and instructions to, a data storage system, at least one input device, and at least one output device. Each computer program may be implemented in a high-level procedural or object-oriented programming language, or in assembly or machine language if desired; and in any case, the language may be a compiled or interpreted language. Suitable processors include, by way of example, both general and special purpose microprocessors. Furthermore, alternate embodiments of the invention that implement the system in hardware, firmware or a combination of both hardware and software, as well as distributing modules and/or data in a different fashion will be apparent to those skilled in the art and are also within the scope of the invention. In addition, it will be obvious to one of ordinary skill to use a conventional database management system such as, by way of non-limiting example, Sybase, Oracle and DB2, as a platform for implementing the present invention. Also, network access devices can comprise a personal computer executing an operating system such as Microsoft Windows™, Unix™, or Apple Mac OS™, as well as software applications, such as a JAVA program or a web browser. Network access devices 203-205 can also be a terminal device, a palm-type computer, mobile WEB access device or other device that can adhere to a point-to-point or network communication protocol such as the Internet protocol. Computers and network access devices can include a processor, RAM and/or ROM memory, a display capability, an input device and hard disk or other relatively permanent storage. Accordingly, other embodiments are within the scope of the following claims.

It will thus be seen that the objects set forth above, among those made apparent from the preceding description, are efficiently attained and, since certain changes may be made in carrying out the above process, in a described product, and in the construction set forth without departing from the spirit and scope of the invention, it is intended that all matter contained in the

above description shown in the accompanying drawing shall be interpreted as illustrative and not in a limiting sense.

It is also to be understood that the following claims are intended to cover all of the generic and specific features of the invention herein described, and all statements of the scope of the invention, which, as a matter of language, might be said to fall therebetween.

10037658.010302

APPENDIX

EXHIBIT A - Code Stubs Generated for Check Address Activity Diagram of FIG. 6

```
package
com.wdr.itops.AcceptanceApplication.Activities.SpawnSubProcessActivities.GenerateConfirmation;

import
com.wdr.itops.infrastructure.ActivityAndDecisionFramework.WorkflowSpawnSubProcessActivity;
import com.wdr.itops.infrastructure.WorkflowInterface.ActivityContext;
import com.wdr.itops.infrastructure.WorkflowInterface.SpawnedWorkflowContext;
import java.sql.Timestamp;
import com.wdr.itops.infrastructure.CommonInfrastructureInterface.CIException;

public class  CheckAddress extends WorkflowSpawnSubProcessActivity
{
 public static String subprocessName() throws CIException
 {
  /************************************************
  Place your code here.
  ************************************************/

  return "checkAddress";
 }

 public static String subprocessPackage() throws CIException
 {
  /************************************************
  Place your code here.
  ************************************************/

  return "GenerateConfirmation";
 }

 public static String subprocessProject() throws CIException
 {
  /************************************************
  Place your code here.
  ************************************************/

  return "AcceptanceApplication";
 }


 public static String projectVersion() throws CIException
 {
  /************************************************
  Place your code here.
  ************************************************/

  return "1";
 }
```

```
 public static void prepareSpawnContext(ActivityContext currentContext, SpawnedWorkflowContext
spawnedContext) throws CIException
  {
   /***********************************************
   Place your code here.
   ***********************************************/

  }
}
```

Exhibit B – Code Segments for Check Address Activity Diagram of FIG. 6

```
public static Hashtable checkAddressDefinition()
  {
    Hashtable workflowSteps = new Hashtable(9, 1);
    Hashtable workflowStep;
    Hashtable workflowDecisions;
    Hashtable workflowDecision;

    workflowSteps.put("Workflow Name", "CheckAddress");
    workflowSteps.put("First Step", "3A15507E03A1");
    workflowSteps.put("Generation Date", "05/12/01 10:22");
    workflowSteps.put("Generator Version", "2.10.8");

    workflowStep = new Hashtable(8, 1);
    workflowStep.put("UniqueName", "3A15507E03A1");
    workflowStep.put("Name", "ValidateAddress");
    workflowStep.put("Type", "Automatic");
    workflowStep.put("Project", "AcceptanceApplication");
    workflowStep.put("PackageSuffix", "Activities.AutomaticActivities");
    workflowStep.put("Package", "CheckAddress");
    workflowStep.put("Decision", "true");
    workflowDecision = new Hashtable(2, 1);
    workflowDecisions = new Hashtable(2, 1);
    workflowDecision.put("Decisions", workflowDecisions);
    workflowDecision.put("Condition Type", "String");
    workflowDecisions.put("Valid", "3A15507E03B5");
    workflowDecisions.put("InValid", "3A15507E03AB");
    workflowStep.put("Next", workflowDecision);
    workflowSteps.put(workflowStep.get("UniqueName"), workflowStep);

    workflowStep = new Hashtable(8, 1);
    workflowStep.put("UniqueName", "3A15507E03B5");
    workflowStep.put("Name", "AddressesValid");
    workflowStep.put("Type", "Automatic");
    workflowStep.put("Project", "AcceptanceApplication");
    workflowStep.put("PackageSuffix", "Activities.AutomaticActivities");
    workflowStep.put("Package", "CheckAddress");
```

```
    workflowStep.put("Decision", "false");
    workflowStep.put("Next", "3A15507E03BF");
    workflowSteps.put(workflowStep.get("UniqueName"), workflowStep);

    workflowStep = new Hashtable(8, 1);
    workflowStep.put("UniqueName", "3A15507E03BF");
    workflowStep.put("Name", "CreateConfirmation");
    workflowStep.put("Type", "ScheduleSubProcess");
    workflowStep.put("Project", "AcceptanceApplication");
    workflowStep.put("PackageSuffix", "Activities.ScheduleSubProcessActivities");
    workflowStep.put("Package", "CheckAddress");
    workflowStep.put("Decision", "false");
    workflowStep.put("Next", "NULL");
    workflowSteps.put(workflowStep.get("UniqueName"), workflowStep);

    workflowStep = new Hashtable(8, 1);
    workflowStep.put("UniqueName", "3A15507E03AB");
    workflowStep.put("Name", "AddressesInvalid");
    workflowStep.put("Type", "Automatic");
    workflowStep.put("Project", "AcceptanceApplication");
    workflowStep.put("PackageSuffix", "Activities.AutomaticActivities");
    workflowStep.put("Package", "CheckAddress");
    workflowStep.put("Decision", "false");
    workflowStep.put("Next", "3A1E48C80214");
    workflowSteps.put(workflowStep.get("UniqueName"), workflowStep);

    workflowStep = new Hashtable(8, 1);
    workflowStep.put("UniqueName", "3A1E48C80214");
    workflowStep.put("Name", "FixInvalidAddress");
    workflowStep.put("Type", "Manual");
    workflowStep.put("Project", "AcceptanceApplication");
    workflowStep.put("PackageSuffix", "Activities.ManualActivities");
    workflowStep.put("Package", "CheckAddress");
    workflowStep.put("Decision", "false");
    workflowStep.put("Next", "3A15507E03A1");
    workflowSteps.put(workflowStep.get("UniqueName"), workflowStep);

    return workflowSteps;
}
```

Exhibit C – Code Segment of User Interface for FixInvalidAddress Manual Activity

```
package com.wdr.itops.AcceptanceApplication.Activities.ManualActivities.CheckAddress;

import com.wdr.itops.infrastructure.WorkflowInterface.ManualActivityContext;
import com.wdr.itops.infrastructure.WorkflowInterface.PrepareIntrayContext;
import com.wdr.itops.infrastructure.WorkflowInterface.PrepareIntrayWorkItem;
import com.wdr.itops.infrastructure.CommonInfrastructureInterface.CIException;
import com.wdr.itops.infrastructure.ActivityAndDecisionFramework.WorkflowManualActivity;
import com.wdr.itops.infrastructure.ManualWorkItem.ManualWorkItem;
import java.util.Vector;
import com.wdr.itops.infrastructure.WorkflowContext.WorkflowContextException;
import com.wdr.itops.infrastructure.ActivityAndDecisionFramework.ActivityAndDecisionException;
```

```
import com.wdr.itops.infrastructure.WorkflowClientInterface.AbstractDetailModel;


public class  FixInvalidAddress extends WorkflowManualActivity
{
 public static void prepareIntray(PrepareIntrayContext workflowContext) throws CIException
 {
  /************************************************
  Place your code here.
  ************************************************/

 }


 public static AbstractDetailModel getModelInfo(Vector workflowContexts) throws CIException
 {
  /************************************************
  Place your code here.
  ************************************************/

  return null;
 }


 public static String manualProceed(String activityClassName, AbstractDetailModel modelInfo, String
userId, ManualActivityContext activityContext) throws CIException
 {
  /************************************************
  Place your code here.
  For now, here is some helpful temporary code:
  ************************************************/

  String result;
  int returnAValue;

  returnAValue = com.sun.java.swing.JOptionPane.showConfirmDialog(null, "Welcome to manual
submit for activity FixInvalidAddress\nDo you want to return a value?");

  if (returnAValue == 0)
     result = com.sun.java.swing.JOptionPane.showInputDialog(null, "Enter an activity return string:
");
  else
     result = "";

  return result;
 }


 public static void addDataToModel(ManualActivityContext workflowContext, AbstractDetailModel
model) throws CIException
 {
  /************************************************
  Place your code here.
  ************************************************/

 }
```

```
 public static AbstractDetailModel createDetailModel(ManualActivityContext workflowContext)
throws CIException
 {
  /************************************************
   Place your code here.
   ************************************************/

   return null;
 }


 public static int getSimilarValue(ManualActivityContext source) throws CIException
 {
  /************************************************
   Place your code here.
   ************************************************/

   return 0;
 }


 public static boolean isManualActivityValid(ManualActivityContext workflowContext) throws
CIException
 {
  /************************************************
   Place your code here.
   ************************************************/

   return true;
 }
}
```

Exhibit D – Code Section to Retrieve Data and Perform Specified Business Logic

```
package com.wdr.itops.AcceptanceApplication.BusinessObjects.Order;

import java.sql.*;
import java.sql.Connection;

import com.wdr.itops.infrastructure.BusinessObjectFramework.*;
import com.wdr.itops.AcceptanceApplication.BusinessObjects.Client.Client;
import com.wdr.itops.AcceptanceApplication.BusinessObjects.Client.ClientKey;
import com.wdr.itops.AcceptanceApplication.BusinessObjects.Instrument.Instrument;
import com.wdr.itops.AcceptanceApplication.BusinessObjects.Instrument.InstrumentKey;
import com.wdr.itops.infrastructure.CommonInfrastructureInterface.CIException;
import com.wdr.itops.AcceptanceApplication.DataTypes.Quantity;
import com.wdr.itops.infrastructure.CICollection.CICollection;
import java.math.BigDecimal;
import com.wdr.itops.infrastructure.DataAccessLayer.SQLBaseClass;
import java.sql.Connection;
```

```
import java.sql.*;
import com.wdr.itops.AcceptanceApplication.DataTypes.OrderCode;
import com.wdr.itops.AcceptanceApplication.DataTypes.ClientCode;
import com.wdr.itops.AcceptanceApplication.DataTypes.InstrumentCode;

public class Order extends BusinessObject
{
	private ClientCode clientCode;
	private InstrumentCode instrumentCode;
	private Quantity quantity;
	private Client orderClient = null;
	private Instrument orderInstrument = null;

	//Derived attribute
	private Integer biggestAllocationQuantity = null;

	private CICollection allocationCollection = new CICollection() ;

public void audit(String message, int Classific)
{
}
/**
* @return boolean
 */
private boolean clearAttributes() {

	clientCode = null;
	instrumentCode = null;
	quantity = null;
	orderClient = null;
	orderInstrument = null;

	allocationCollection = new CICollection() ;
	return true;
}
	public CICollection createAllocationCollection() throws CIException
	{
		// run the initial query to populate the collection
		getBusinessObjectFactory().runPersistenceMethod(getKey(),
"createAllocationCollection");

		return allocationCollection;
	}
/**
*/
public void doSomeDynamicSQL()
{
	final Connection con = SQLBaseClass.getConnection();
	String clientCode = (getClientCode()).toString();
	final String query = "select client_code from aa_client where client_code =" + clientCode;
	Statement stmt = null;
	try
	{
		stmt = con.createStatement();
		final ResultSet rs = stmt.executeQuery(query);
		while (rs.next())
```

```
                {
                        System.out.println(rs.getString(1));
                }
        }
        catch (SQLException s)
        {
                System.out.println(s.getErrorCode() + s.getMessage());
        }
                finally
                {
                        try{
                        stmt.close();
                }
                        catch(SQLException sq){
                        }
                }

}
/**
* @param businessObject com.wdr.itops.infrastructure.BusinessObjectFramework.BusinessObject
 */
public void doSomeDynamicSQLUsingPreparedStatements(BusinessObject businessObject) {

        Timestamp timeStamp = businessObject.getOptimisticLockTimeStamp();
        final Connection con = SQLBaseClass.getConnection();
        final String query = "select allocation_number from aa_allocation where update_timestamp
<?";

        PreparedStatement stmt = null;
        try{
         stmt =  con.prepareStatement(query);
         stmt.setTimestamp(1, timeStamp);
         ResultSet rs = stmt.executeQuery();
         while(rs.next()){
                 System.out.println(rs.getInt("allocation_Number"));
         }
         stmt.close();
        }
        catch(SQLException s){
                System.out.println(s.getMessage());
        }
        }
        public CICollection getAllocationCollection() throws CIException
        {
                return allocationCollection;
        }

        public CICollection getAllocationCollectionFromDatabase() throws CIException
        {
                if (allocationCollection.isEmpty())
                {
                        // populate the collection
                        getBusinessObjectFactory().runPersistenceMethod(getKey(),
"fetchAllocationCollection");
                }
```

```
                return allocationCollection;
        }

/**
* @return java.lang.Integer
 */
public java.lang.Integer getBiggestAllocationQuantity() {
        return biggestAllocationQuantity;
}
/**
*/
public int getCacheHint() {

        return RECYCLE;

        }
/**
* @return java.lang.String
 */
public Client getClient() throws CIException
{
        if (orderClient==null)
                orderClient = (Client) getBusinessObjectFactory().tryRetrieve(new
ClientKey(getClientCode() ) );

        return orderClient;
}
/**
* @return java.lang.String
 */
public ClientCode getClientCode()
{
        return clientCode;
}
        /**
        / Determines order of flushing -
        / Order(3),Allocation(2),Confirmation(1) then everything else
        */
        protected int getDependency()
        {
                return 3;
        }
/**

 * @return java.lang.String
 */
public Instrument getInstrument() throws CIException
{
        if (orderInstrument==null)
                orderInstrument = (Instrument) getBusinessObjectFactory().tryRetrieve(new
InstrumentKey(getInstrumentCode()));

        return orderInstrument;
}
/**
* @return java.lang.String
```

```
 */
public InstrumentCode getInstrumentCode()
{
	return instrumentCode;
}
/**
 * Gets the order code from the key and returns it.
* @return java.lang.String
 */
public OrderCode getOrderCode()
{
	return ((OrderKey) getKey()).getOrderCode();
}
/**
* @return java.lang.String
 */
public Quantity getQuantity()
{
	return quantity;
}
/**
* @return boolean
 */

public boolean resetObject()
{
	boolean answer = super.resetObject(true);
	if (answer)
	{
		answer = clearAttributes();
	}
	return answer;
}
public void setAllocationCollection(CICollection newAllocationCollection)
{
	allocationCollection = newAllocationCollection;
	touch();
}
/**
* @param newBiggestAllocationQuantity java.lang.Integer
 */
public void setBiggestAllocationQuantity(java.lang.Integer newBiggestAllocationQuantity) {
	biggestAllocationQuantity = newBiggestAllocationQuantity;
}
/**
* @param s java.lang.String
 */
public void setClientCode(ClientCode c)
{
	orderClient = null;
	clientCode = c;
	touch();
}
/**
* @param s java.lang.String
 */
```

```
public void setInstrumentCode(InstrumentCode i)
{
        orderInstrument = null;
        instrumentCode = i;
        touch();
}
/**
* @param s java.lang.String
 */
public void setQuantity(Quantity s)
{
        quantity = s;
        touch();
}
}
```

```
package com.wdr.itops.AcceptanceApplication.BusinessObjects.Order;


import com.wdr.itops.infrastructure.BusinessObjectFramework.*;
import com.wdr.itops.infrastructure.CommonInfrastructureInterface.CIException;
import com.wdr.itops.infrastructure.KeyFramework.Key;
import com.wdr.itops.infrastructure.KeyFramework.FullPersistenceRR;
import com.wdr.itops.AcceptanceApplication.DataTypes.OrderCode;

public class OrderKey extends Key
{
        private OrderCode orderCode;
/**
 * Insert the method's description here.
*/
public OrderKey() throws CIException
{
}
/**
 * Creates the OrderKey based on an OrderCode that has been passed in
* @param orderCode java.lang.String
 */
public OrderKey(OrderCode o) throws CIException
{
        orderCode = o;
}
/**
 * deSerialize method comment.
 */
protected void deSerialize() throws CIException

{
        String orderCodeAsString = getNextKeyComponentAsString();
        orderCode = new OrderCode(orderCodeAsString);
}
/**
 * Insert the method's description here.
```

```
* @return java.lang.String
 */
public OrderCode getOrderCode()

	{
		return orderCode;
	}
/**
 * serialize method comment.
 */
protected void serialize()

	{
		setNextKeyComponent(getOrderCode());
	}
}
```

Exhibit E – Example of Workflow Routing Class for Invoking Workflow Application

```
package com.wdr.itops.infrastructure.WorkflowFramework;

import com.wdr.itops.infrastructure.WorkflowContext.WorkflowContext;
import com.wdr.itops.infrastructure.DataTypes.CIDateTime;
import com.wdr.itops.infrastructure.GenericInfrastructureContainer.GenericIContainer;
import com.wdr.itops.infrastructure.CommonInfrastructureInterface.CIException;
import com.wdr.itops.infrastructure.CommonInfrastructureBase.CIFatalException;
import com.wdr.itops.infrastructure.CommonInfrastructureBase.CINonFatalException;
import java.util.Hashtable;
import java.lang.reflect.*;
public class GenericWorkflowRouting
{
public static Class findClass(String className) throws WorkflowNonFatalException
{
	Class aClass;
	try
	{
		aClass = Class.forName(className);
	}
	catch (ClassNotFoundException anException)
	{
		throw new WorkflowNonFatalException("GenericWorkflowRouting", "findClass",
"WORKFLOW: ClassNotFoundException trying to resolve class " + className);
	}
	return aClass;
}

public static Class findClassOrNull(String className)
{
	Class aClass = null;
	try
	{
		aClass = Class.forName(className);
```

```
        }
        catch (ClassNotFoundException anException)
        {
        }
        return aClass;
}
public static Method findMethod(Class aClass, String methodName, Class[] paramTypes) throws
WorkflowNonFatalException
{
        Method result = null;
        try
        {
                result  = aClass.getMethod(methodName, paramTypes);
        }
        catch (NoSuchMethodException anException)
        {
                throw new WorkflowNonFatalException("GenericWorkflowRouting", "findMethod",
"WORKFLOW: NoSuchMethodException trying to resolve " + methodName
                        + " for class " + aClass.getName());
        }
        return result;
}

public static Method findMethodOrNull(Class aClass, String methodName, Class[] paramTypes)
{
        Method result = null;
        try
        {
                result  = aClass.getMethod(methodName, paramTypes);
        }
        catch (NoSuchMethodException anException)
        {}
        return result;
}

private static CIException invocationTargetException(Throwable caughtException, String
caughtExceptionName, String localMethodName, String executionMethodName)
{
        CIException result = null;
        String details = "Caught " + caughtExceptionName + ": " + caughtException.getMessage() +
", while trying to invoke " + executionMethodName;
        // CIFatals => WorkflowFatalException
        if (caughtException instanceof CIFatalException)
        { // can cast to exception as CIFatalException extends exception
                result = new WorkflowFatalException(caughtException,
"GenericWorkflowRouting", localMethodName, details);
        }
        else
        { // all other Exceptions (Java and CINon-Fatal) are thrown up as non-fatals
                result = new WorkflowNonFatalException(caughtException,
"GenericWorkflowRouting", localMethodName, details);
        }
        return result;
}
/**
 * Insert the method's description here.
```

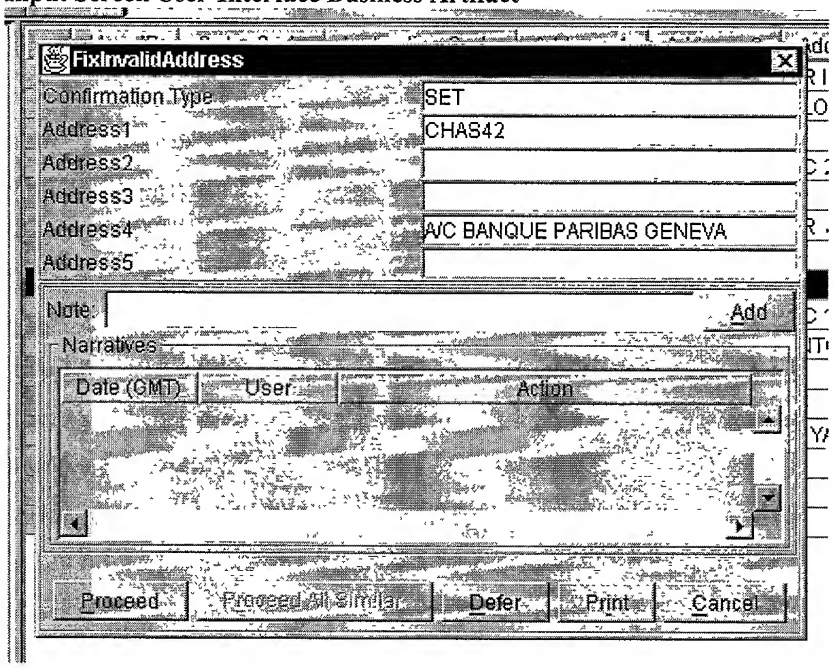
```
*/
public static Object invokeInstanceMethod(Method executionMethod, Object anInstance, Object[]
params) throws CIException
{
	try
	{
		return executionMethod.invoke(anInstance, params);
	}
	catch(IllegalAccessException anException)
	{
		throw reflectionException(anException,"IllegalAccessException",
"invokeWFMethodOfTypeBoolean", executionMethod.getName());
	}
	catch (InvocationTargetException anException)
	{
		throw invocationTargetException(anException.getTargetException(),
anException.getTargetException().getClass().getName(), "invokeWFMethodOfTypeBoolean",
executionMethod.getName());
	}

}
/**
 * Insert the method's description here.
*/
public static Object invokeStaticWFMethod(Method executionMethod, Class[] paramTypes, Object[]
params) throws CIException
{
	try
	{
		Object result = executionMethod.invoke(paramTypes, params);
		GenericIContainer.getExceptionService().throwPostponedExceptions();
		return result;
	}
	catch (IllegalAccessException anException)
	{
		throw reflectionException(anException, "IllegalAccessException",
"invokeWFMethod", executionMethod.getName());
	}
	catch (InvocationTargetException anException)
	{
		throw invocationTargetException(anException.getTargetException(),
anException.getTargetException().getClass().getName(), "invokeWFMethod",
executionMethod.getName());
	}
}
private static WorkflowNonFatalException reflectionException(Throwable caughtException, String
caughtExceptionName, String localMethodName, String executionMethodName)
{
	return new WorkflowNonFatalException(caughtException, "GenericWorkflowRouting",
localMethodName, "Caught " + caughtExceptionName + ": " + caughtException.getMessage() + ",
while trying to invoke " + executionMethodName);
}

}
```

Exhibit F – Example of User-Interface, Business Object and Data Artifacts for a Web-Based Application

Input Screen User-Interface Business Artifact



Attributes of Instruction Business Object Business Artifact

```
public class Instruction extends BusinessObject
{
        private ConfType confirmationType;
        private String addressOne;
        private String addressTwo;
        private String addressThree;
        private String addressFour;
        private String addressFive;
}
```

Data Artifacts of Instruction Data model

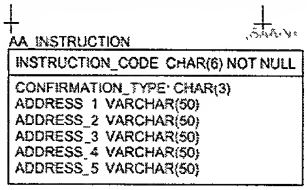
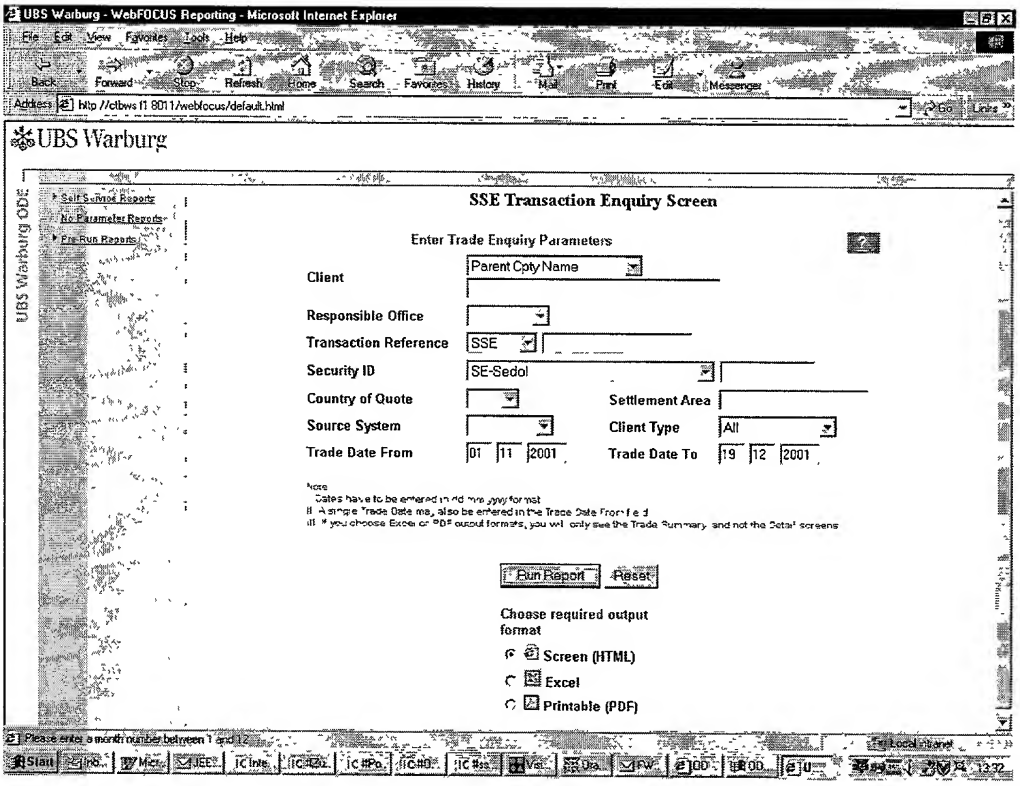


Exhibit G – Example of User-Interface and Data Artifacts for a Reporting Application

Query Screen User-Interface Business Artifact



Report Results Screen Including Responsive Data

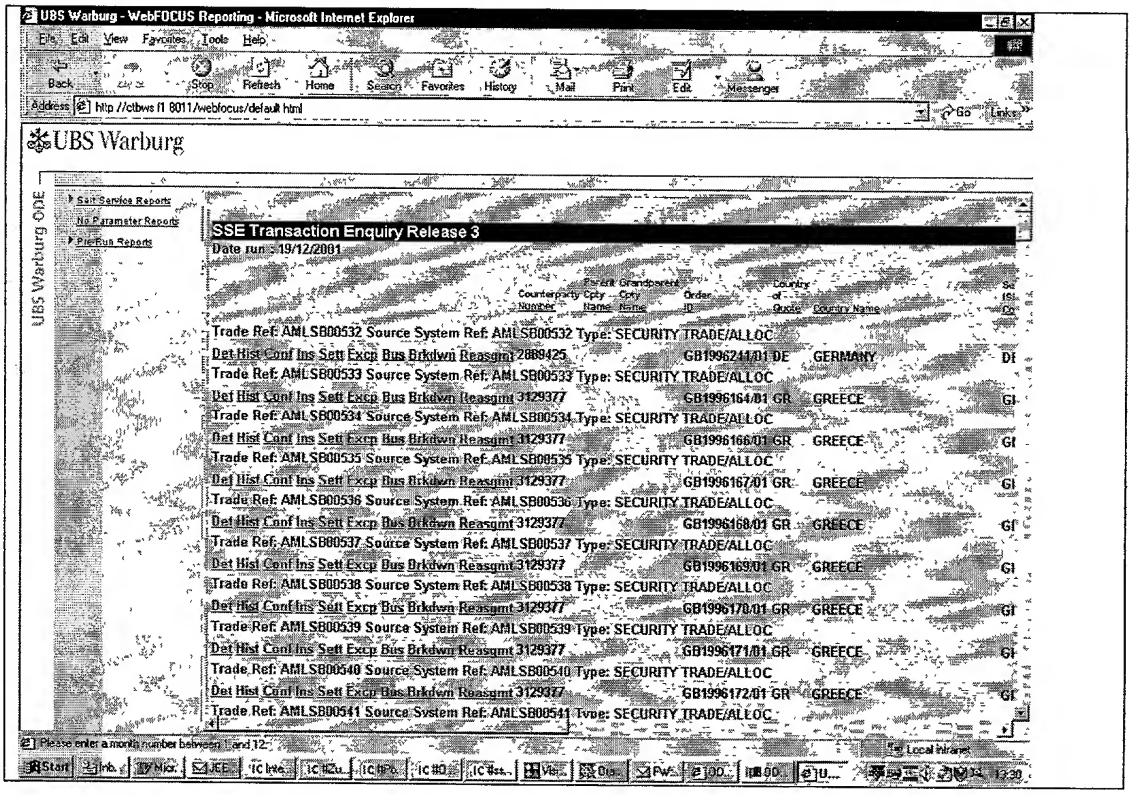

UBS Warburg - WebFOCUS Reporting - Microsoft Internet Explorer
UBS Warburg
UBS Warburg ODE
Self-Service Reports
No Parameter Reports
Previous Reports
SSE Transaction Enquiry Release 3
Date run : 19/12/2001
Trade Ref: AMLSB00532 Source System Ref: AMLSB00532 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00533 Source System Ref: AMLSB00533 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00534 Source System Ref: AMLSB00534 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00535 Source System Ref: AMLSB00535 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00536 Source System Ref: AMLSB00536 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00537 Source System Ref: AMLSB00537 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00538 Source System Ref: AMLSB00538 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00539 Source System Ref: AMLSB00539 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00540 Source System Ref: AMLSB00540 Type: SECURITY TRADE/ALLOC
Trade Ref: AMLSB00541 Source System Ref: AMLSB00541 Type: SECURITY TRADE/ALLOC

Exhibit H – Example of Activity, Business Object and Data Artifacts for a Service Request Application

Activity and Business Object Business Artifacts for Configuration Service

```
package com.wdr.itops.infrastructure.ConfigurationService;

/**
 * ConfigurationService.java
 *
*/
import javax.naming.Name;
import javax.naming.NamingException;
import javax.naming.NamingEnumeration;
import java.util.Hashtable;
public interface ConfigurationService extends ConfigConstants
{
public Object addToEnvironment(String propName, Object propVal) throws NamingException;


/**
 * Binds a name to an object.
 */
public void bind(String arg1, Object arg2) throws NamingException;
/**
 * Binds a name to an object.
 */
public void bind(Name arg1, Object arg2) throws NamingException ;
/**
 *  Composes the name of this context with a name relative to this context.
 */
public String composeName(String arg1, String arg2) throws NamingException ;
/**
 *  Composes the name of this context with a name relative to this context.
 */
public Name composeName(Name arg1, Name arg2) throws NamingException;
/**
 * Creates and binds a new context.
 */
public ConfigurationService createSubcontext(String arg1) throws NamingException;
/**
 * Creates and binds a new context.
 */
public ConfigurationService createSubcontext(Name arg1) throws NamingException ;
/**
 *  Destroys the named context and removes it from the namespace.
 */
public void destroySubcontext(String arg1) throws NamingException;
/**
 *  Destroys the named context and removes it from the namespace.
 */
public void destroySubcontext(Name arg1) throws NamingException;
public Hashtable getEnvironment() throws NamingException;
public abstract NamingEnumeration list(String name) throws NamingException;
public abstract NamingEnumeration list(Name name) throws NamingException;
public abstract NamingEnumeration listBindings(String name) throws NamingException;
```

```
public abstract NamingEnumeration listBindings(Name name) throws NamingException;
/**
 * Retrieves the named object.
 */
public Object lookup(String arg1) throws NamingException ;
/**
 * Retrieves the named object.
 */
public Object lookup(Name arg1) throws NamingException ;
/**
 * Binds a name to an object, overwriting any existing binding.
 */
public void rebind(String arg1, Object arg2) throws NamingException;
/**
 * Binds a name to an object, overwriting any existing binding.
 */
public void rebind(Name arg1, Object arg2) throws NamingException;
public Object removeFromEnvironment(String propName) throws NamingException;
/**
 *  Unbinds the named object.
 */
public void unbind(String arg1) throws NamingException;
/**
 *  Unbinds the named object.
 */
public void unbind(Name arg1) throws NamingException;
}
```

Data Artifacts for Configuration Service Application